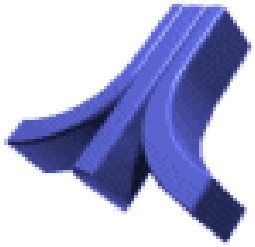
The NetBSD Project

"Of course it runs NetBSD"



OSC2011神戸 日欧68030魂 コラボ展示 おまけ企画

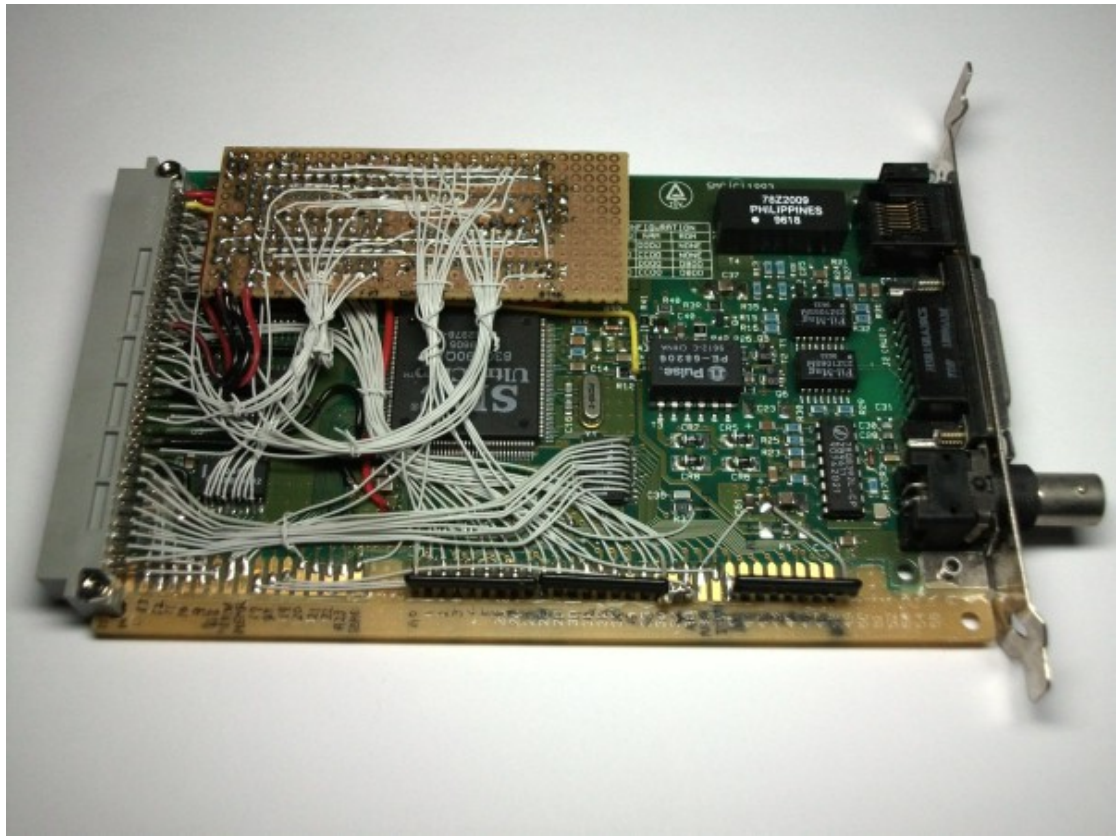
「ATARI用NICアダプタ“EtherNEC”とNetBSD bus_space API実装」

Izumi Tsutsui

tsutsui@NetBSD.org

OSC 2010 Kansai@Kobe

ATARI TT030用 SMC_TT製作展示 と 実機デモ



展示デモURL http://www.ceres.dti.ne.jp/tsutsui/smc_tt/SMC_TT-OSC2010Kobe.gif

ATARI TT030 & SMC_TT

- **ATARI TT030 とは？**
  - 1990年発売 MC68030 32MHz 搭載 32ビットパソコン
  - 当時としては最高レベルのグラフィックとサウンド
  - 現在でも一部に熱狂的ファンユーザーが存在
  - 後継機（Falcon）には 68060アクセラレータやPCIブリッジも

    参考URL http://www.powerphenix.com/CT60/english/overview63.htm

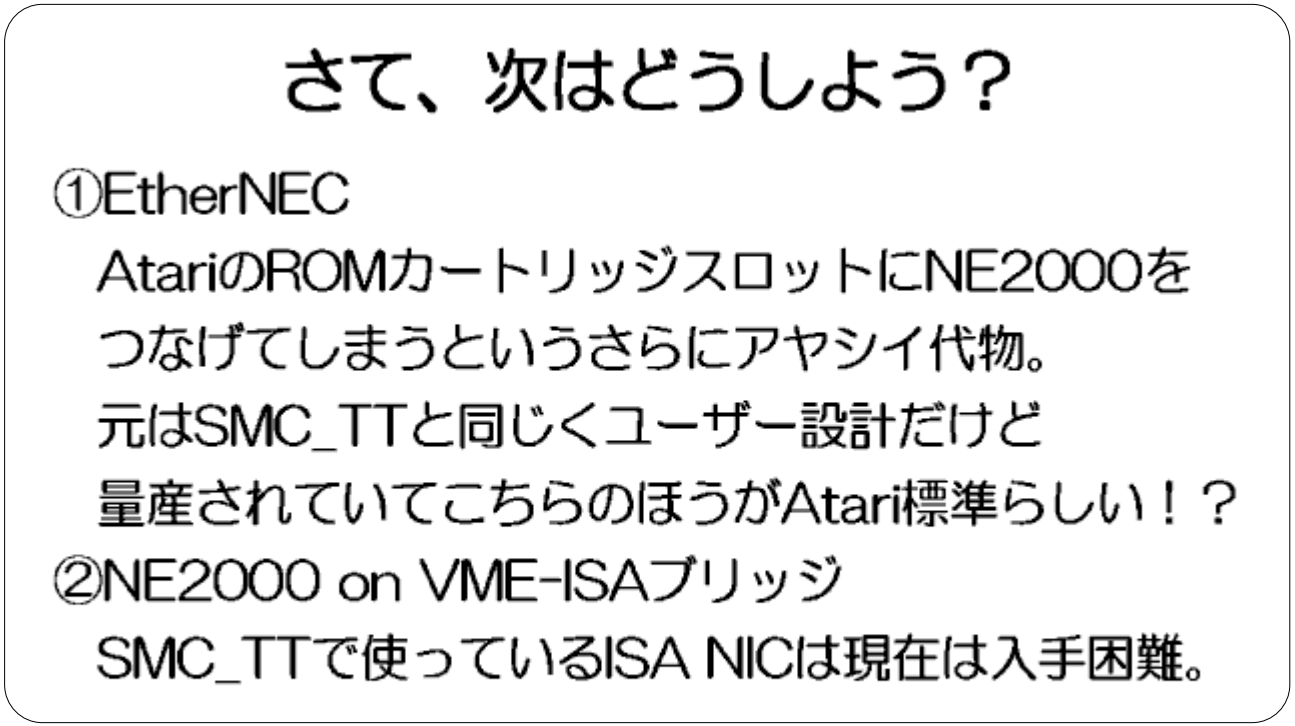
    ・どこかのマシンで似たようなマシンの話を聞いたような……

- **SMC_TT とは？**
  - SMC社製 ISA用NICにブリッジ回路を付けて
    むりやり ATARI TT030 のVMEバスに挿すNICアダプタ

    ・これもどこかのマシンで似たようなNICの話を聞いたような……

もう一つのATARI用NIC - EtherNEC

OSC2010神戸展示 最後の次回予告(笑)スライド

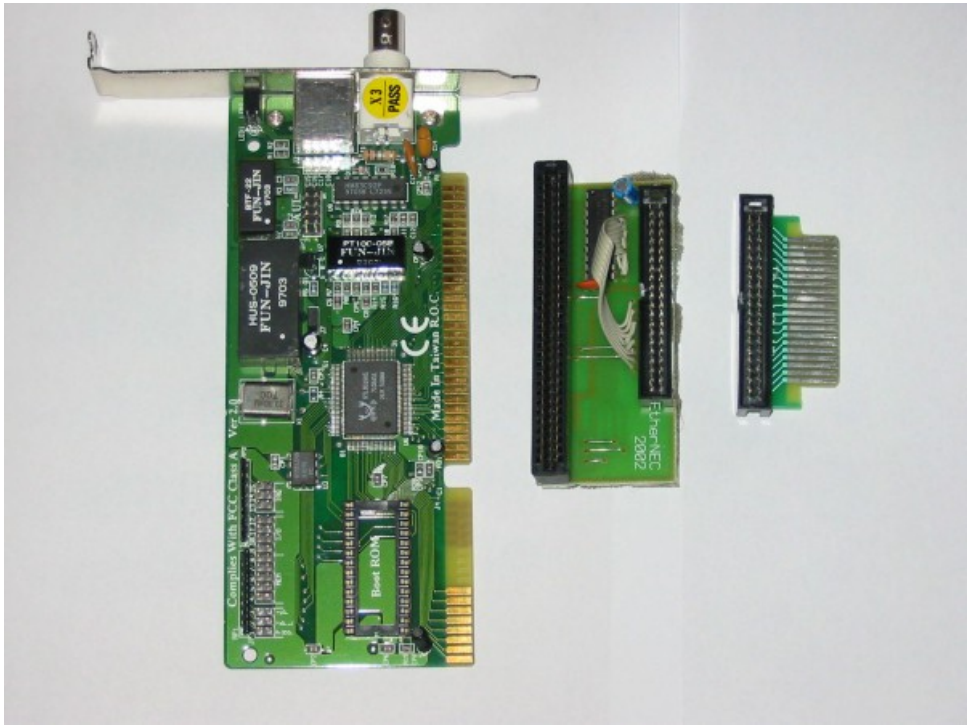


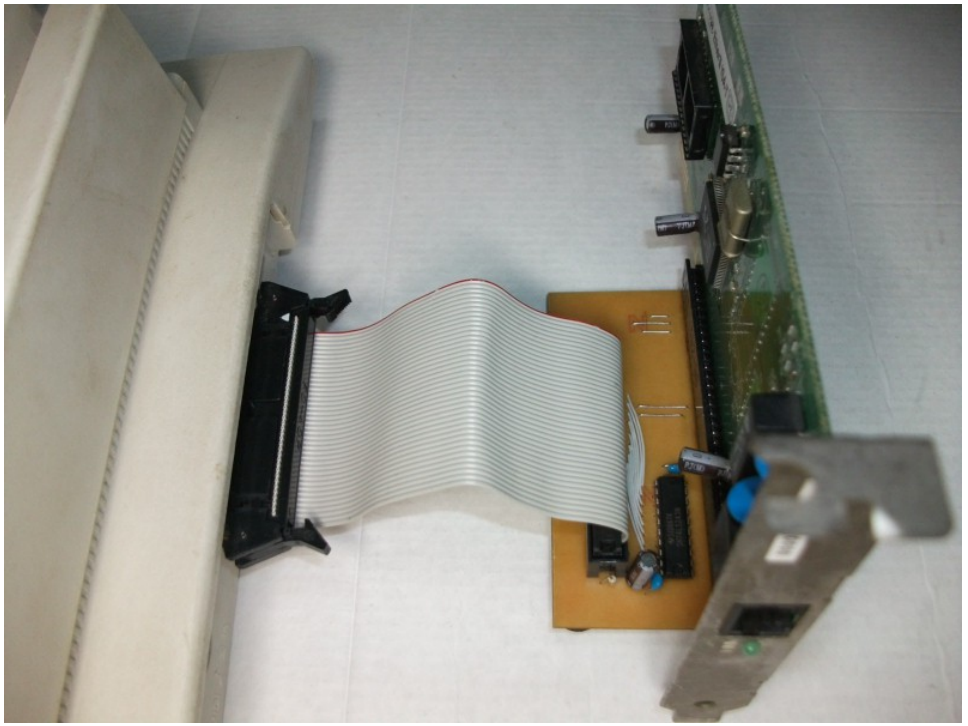
- SMC_TTは作成の敷居が高すぎ (GALライタも必要)
- 使用するSMC製NICも入手困難

……そんなあなたにEtherNEC

EtherNECとは

- ISA NE2000⇔ATARI ROMカートリッジスロット接続アダプタ　http://hardware.atari.org/ether/





- TOS（ATARI純正OS）用ドライバもあり
  http://home.arcor.de/thomas.redelberger/prj/atari/etherne/

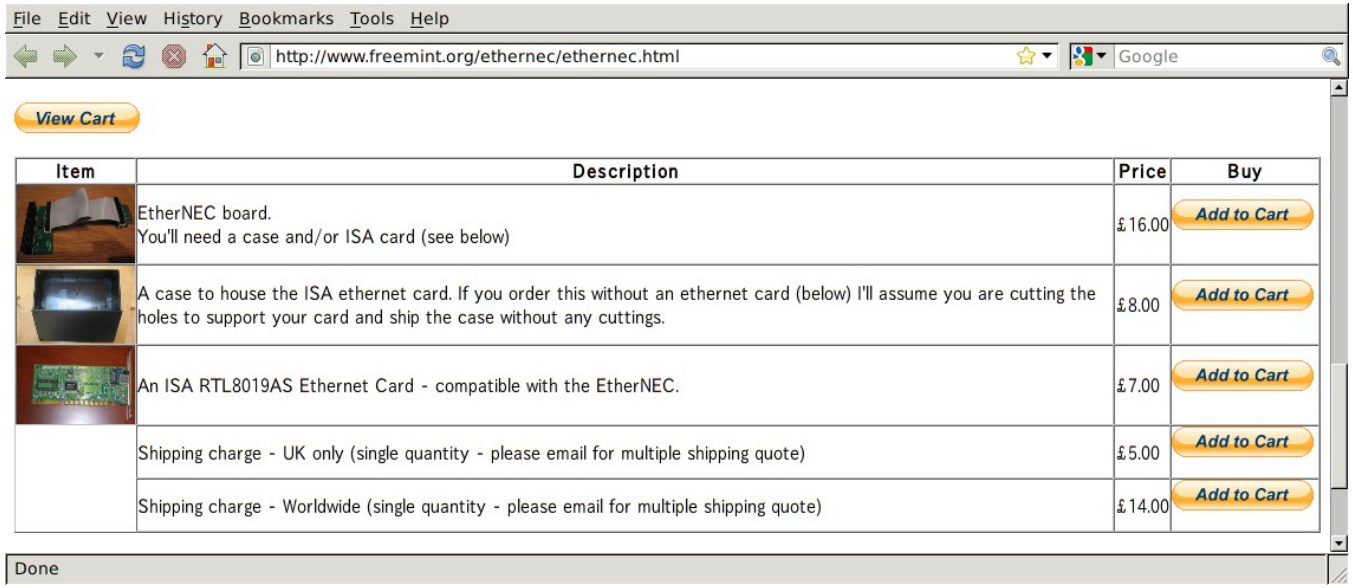
ATARI界におけるEtherNEC

- ドイツのヘビー ATARI ユーザ曰く:

  “The most used and most available network solution on Atari.”

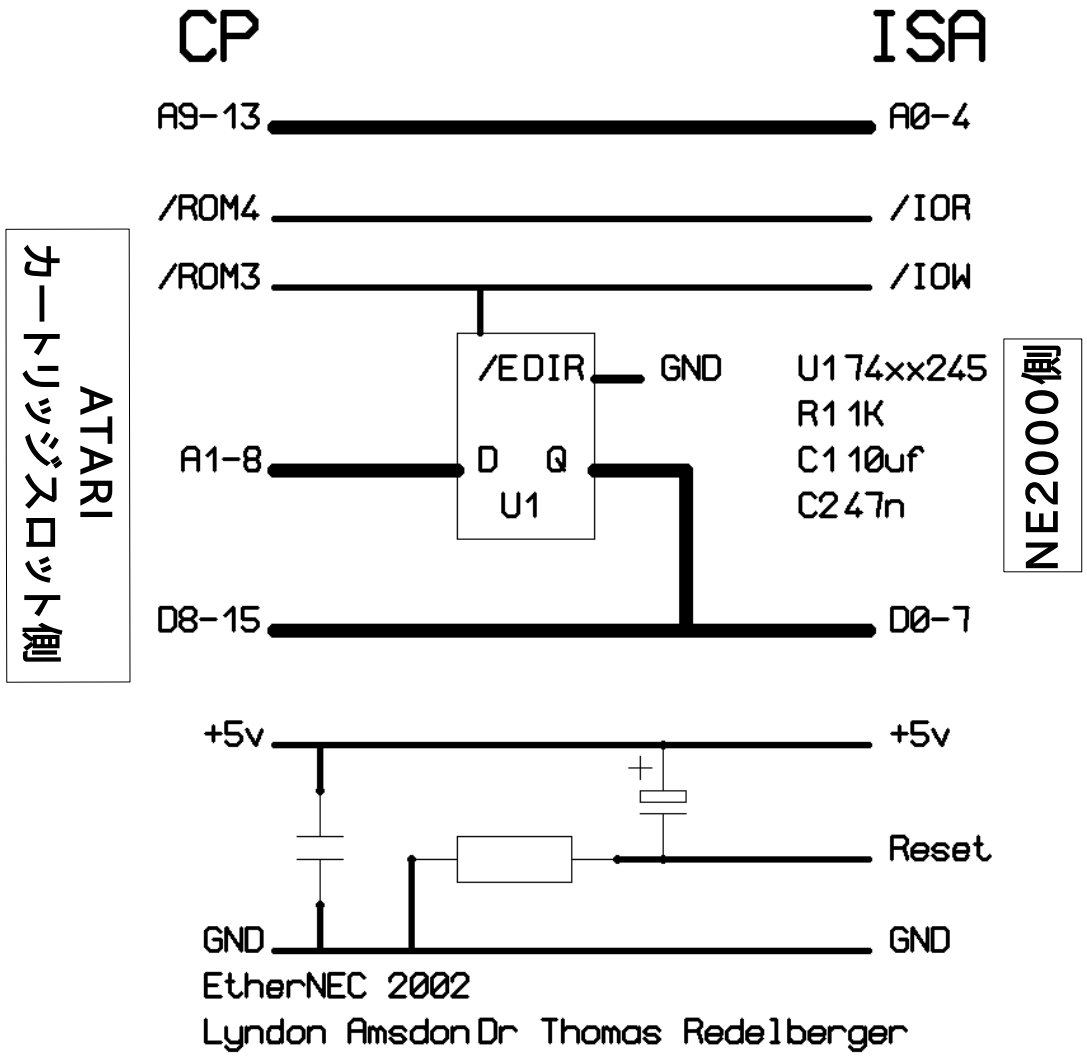
- 2011年4月現在も 通販ページあり。海外発送可。

  http://www.freemint.org/ethernec/ethernec.html



EtherNEC 回路図

http://home.arcor.de/thomas.redelberger/prj/atari/etherne/



- あまりに簡単すぎて
  逆に理解が難しい!?

- そもそもつなぐ相手の
  NE2000ってどんな仕様!?

NE2000 のレジスタ構成

http://realtek.info/pdf/rtl8019as.pdf (互換チップ RTL8019AS データシート)

内部に32個分のレジスタが存在

5.1.1. Register Table

No (Hex)	Page0		Page1	Page2	Page3	
	[R]	[W]	[R/W]	[R]	[R]	[W]
00	CR	CR	CR	CR	CR	CR
01	CLDA0	PSTART	PAR0	PSTART	*9346CR*	*9346CR*
02	CLDA1	PSTOP	PAR1	PSTOP	*BPAGE*	*BPAGE*
03	BNRY	BNRY	PAR2	-	*CONFIG0*	-
04	TSR	TPSR	PAR3	TPSR	*CONFIG1*	*CONFIG1*
05	NCR	TBCR0	PAR4	-	*CONFIG2*	*CONFIG2*
06	FIFO	TBCR1	PAR5	-	*CONFIG3*	*CONFIG3*
07	ISR	ISR	CURR	-	-	*TEST*
08	CRDA0	RSAR0	MAR0	-	*CSNSAV*	-
09	CRDA1	RSAR1	MAR1	-	-	*HLTCLK*
0A	*8019ID0*	RBCR0	MAR2	-	-	-
0B	*8019ID1*	RBCR1	MAR3	-	*INTR*	-
0C	RSR	RCR	MAR4	RCR	-	*FMWP*
0D	CNTR0	TCR	MAR5	TCR	*CONFIG4*	-
0E	CNTR1	DCR	MAR6	DCR	-	-
0F	CNTR2	IMR	MAR7	IMR	-	-
10-17	Remote DMA Port					
18-1F	Reset Port					

NE2000 のレジスタ構成

http://realtek.info/pdf/rtl8019as.pdf (互換チップ RTL8019AS データシート)

内部に32個分のレジスタが存在

5.1.1. Register Table

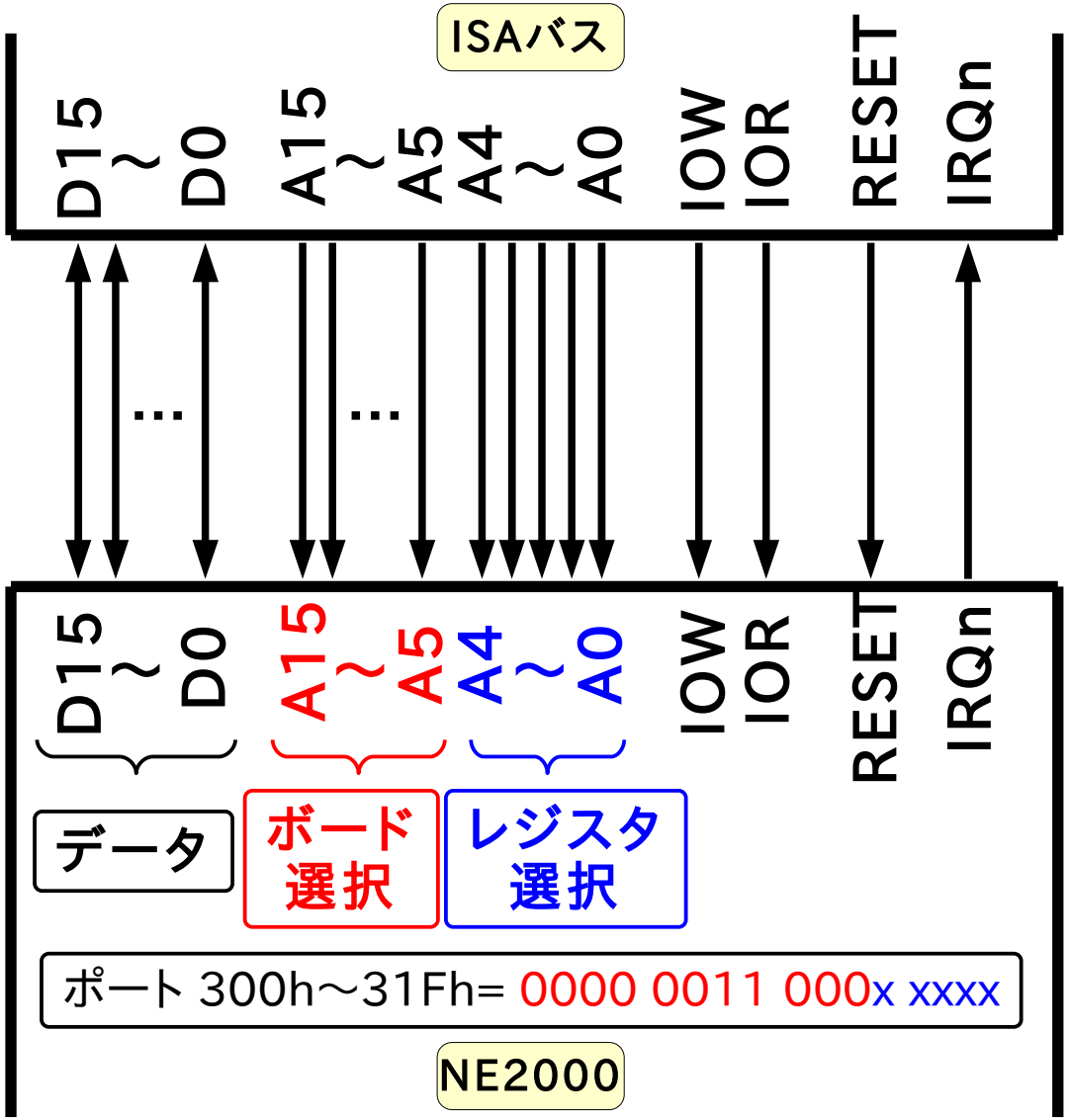
No (Hex)	Page0		Page1	Page2	Page3	
	[R]	[W]	[R/W]	[R]	[R]	[W]
00						
01						
02						
03						
04						
05						
06						
07						
08						
09						
0A						
0B						
0C						
0D						
0E						
0F						
10-17						
18-1F						

- 例えば ISAバスなら
  0x280～0x29F　あるいは
  0x300～0x31F　といった
  計32番地でアクセス

- パケット送受信バッファ(16KB)も
  これらのレジスタ経由でアクセス

- これら 計32個分のレジスタに
  適当に読み書きして パケット送受信

ISAバスの NE2000接続

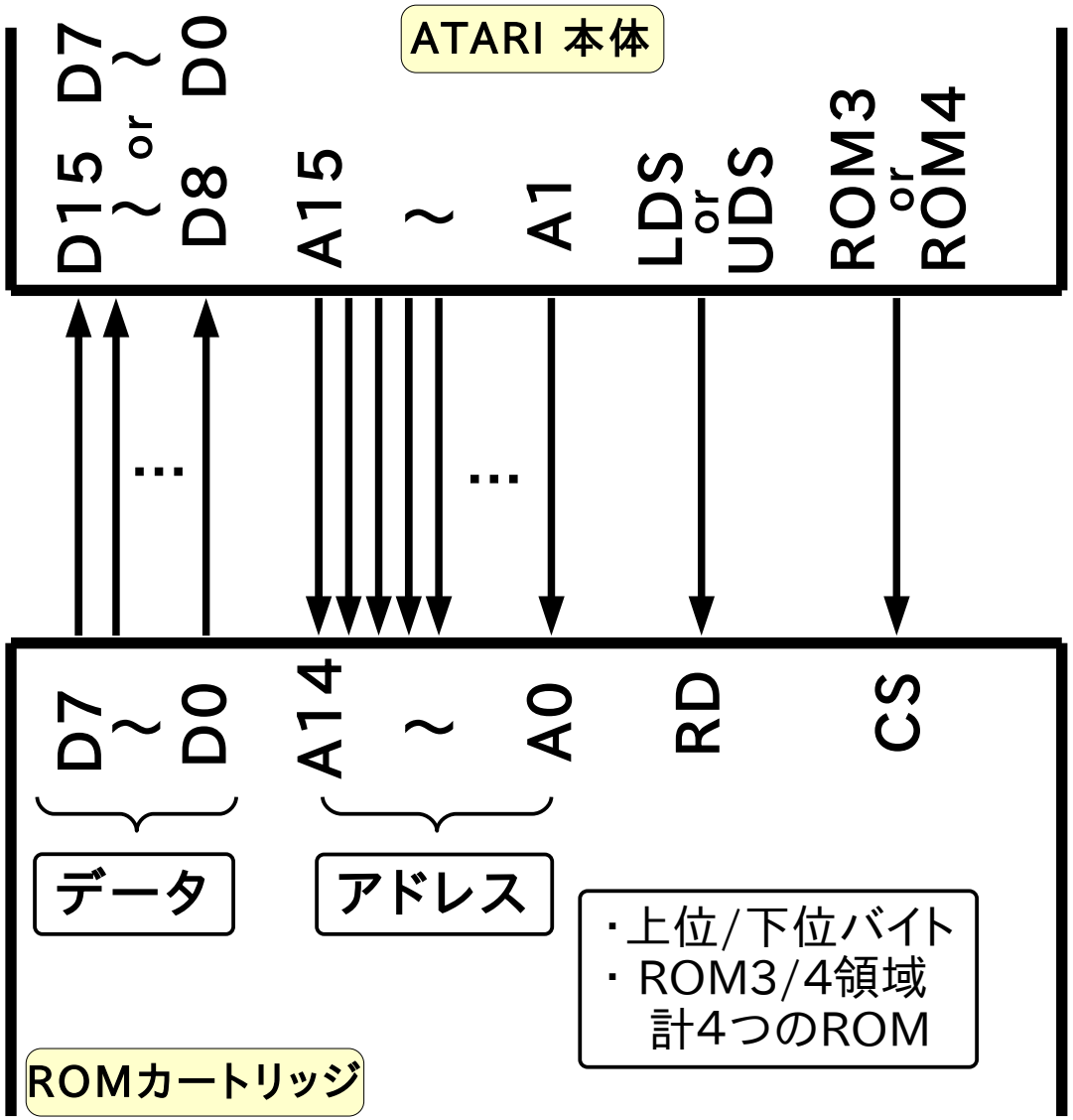


信号線概要

- D15～0: データ(16bit mode)
- D7～0: データ (8bit mode)
- A15～5: ボード選択
- A0～4: レジスタ選択
- IOW: 書き込み要求
- IOR: 読み出し要求
- RESET: リセット信号
- IRQn: 割り込み要求

ATARI ROMカートリッジ接続信号

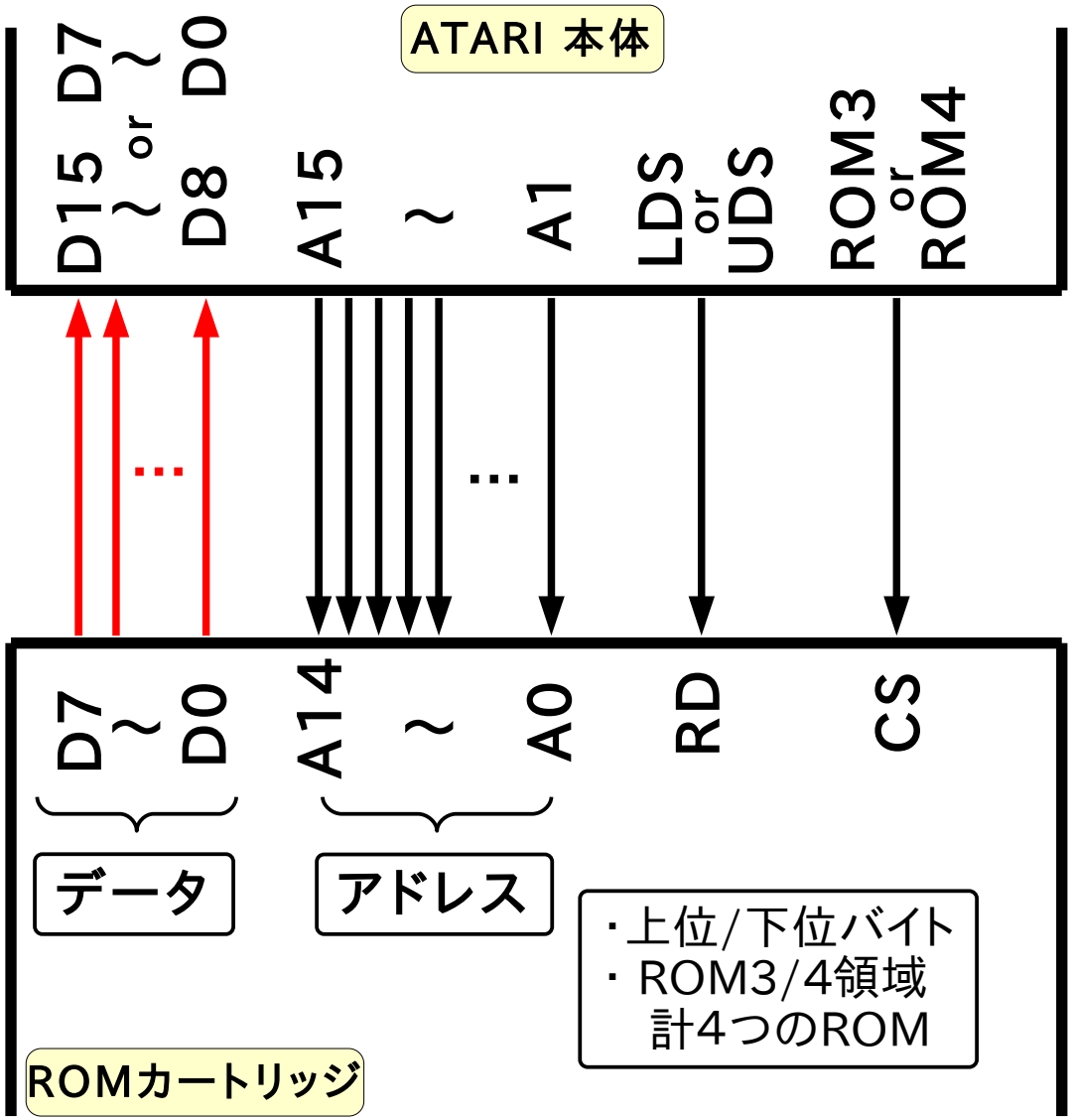
http://wiki.newtosworld.de/Cartridge



信号線概要

- D15～8: データ (odd)
- D7～0: データ (even)
- A15～1: アドレス
- LDS: 下位データ選択
- UDS: 上位データ選択
- ROM3: FB0000h 領域
- ROM4: FA0000h 領域
- RD: 読み出し要求
- CS: チップ選択

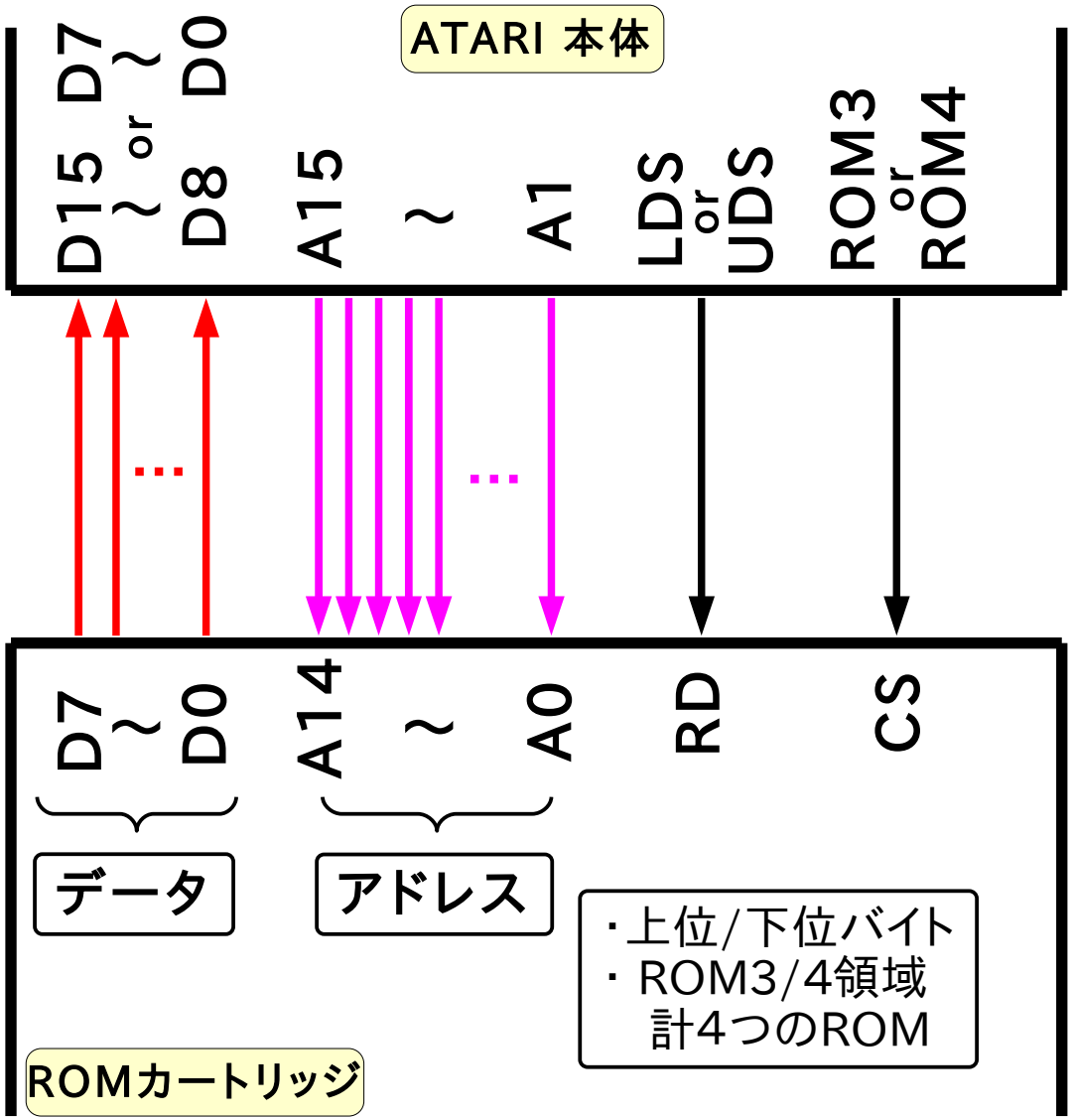
How to connect NE2000 to ROM slot? (1)



- ROMの場合は全て直結
- 文字通り Read Only なので
  データ線は上向き一方通行

⇒NE2000を接続する場合
書き込みはどうするの?

How to connect NE2000 to ROM slot? (1)

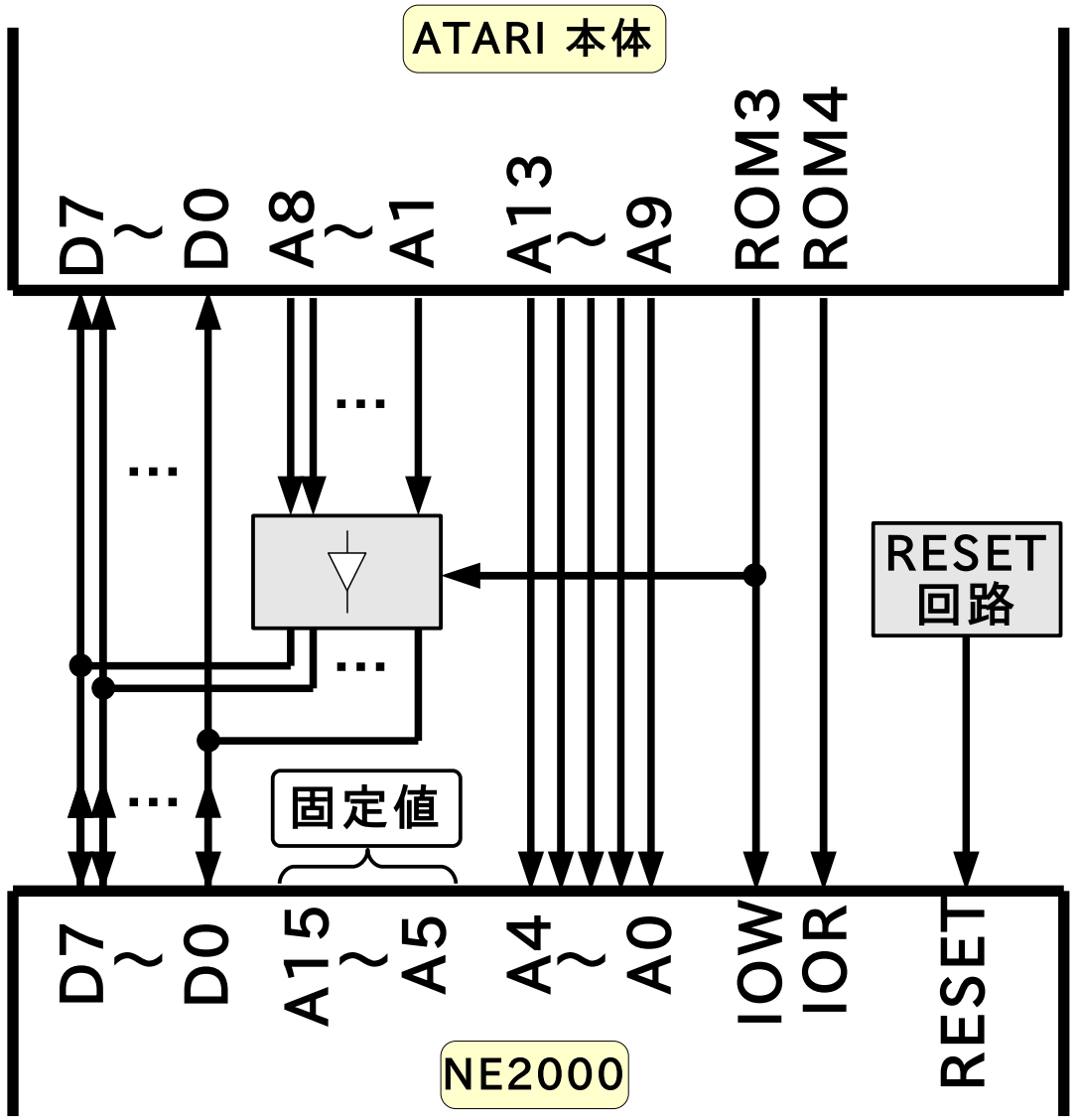


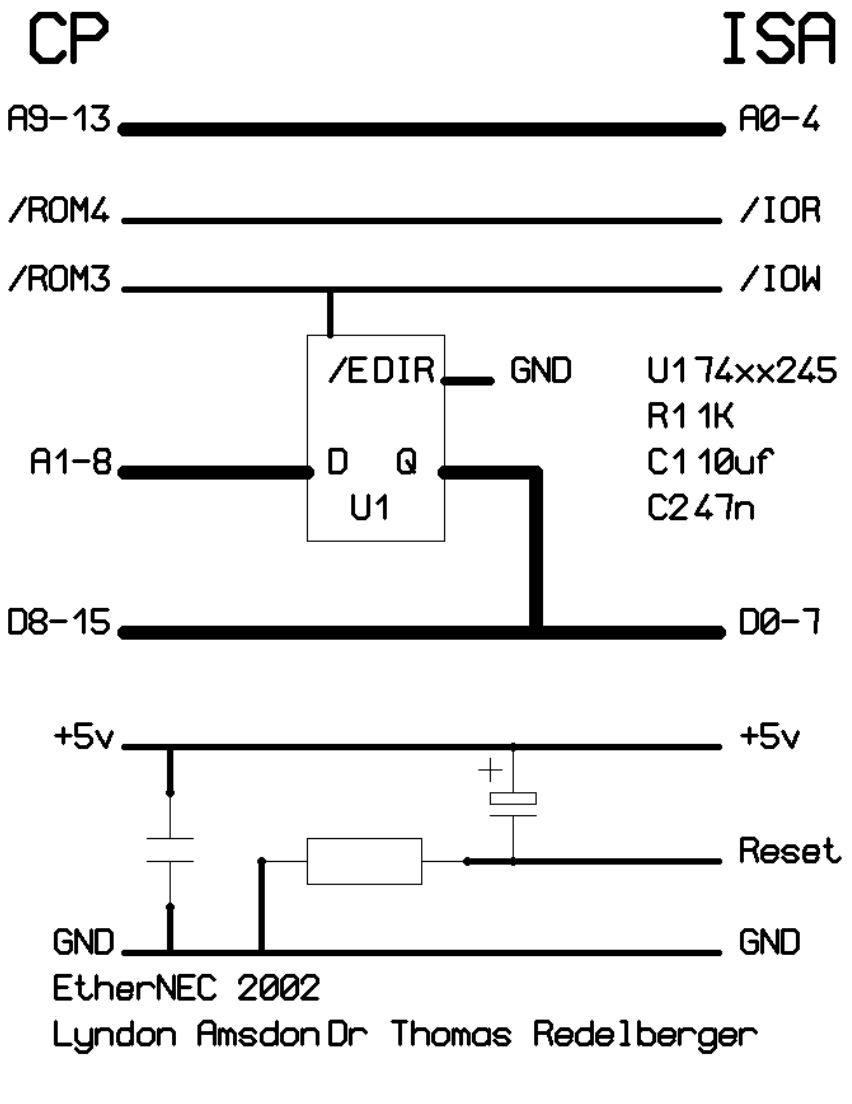
- ROMの場合は全て直結
- 文字通り Read Only なので
  データ線は上向き一方通行

⇒NE2000を接続する場合
書き込みはどうするの?

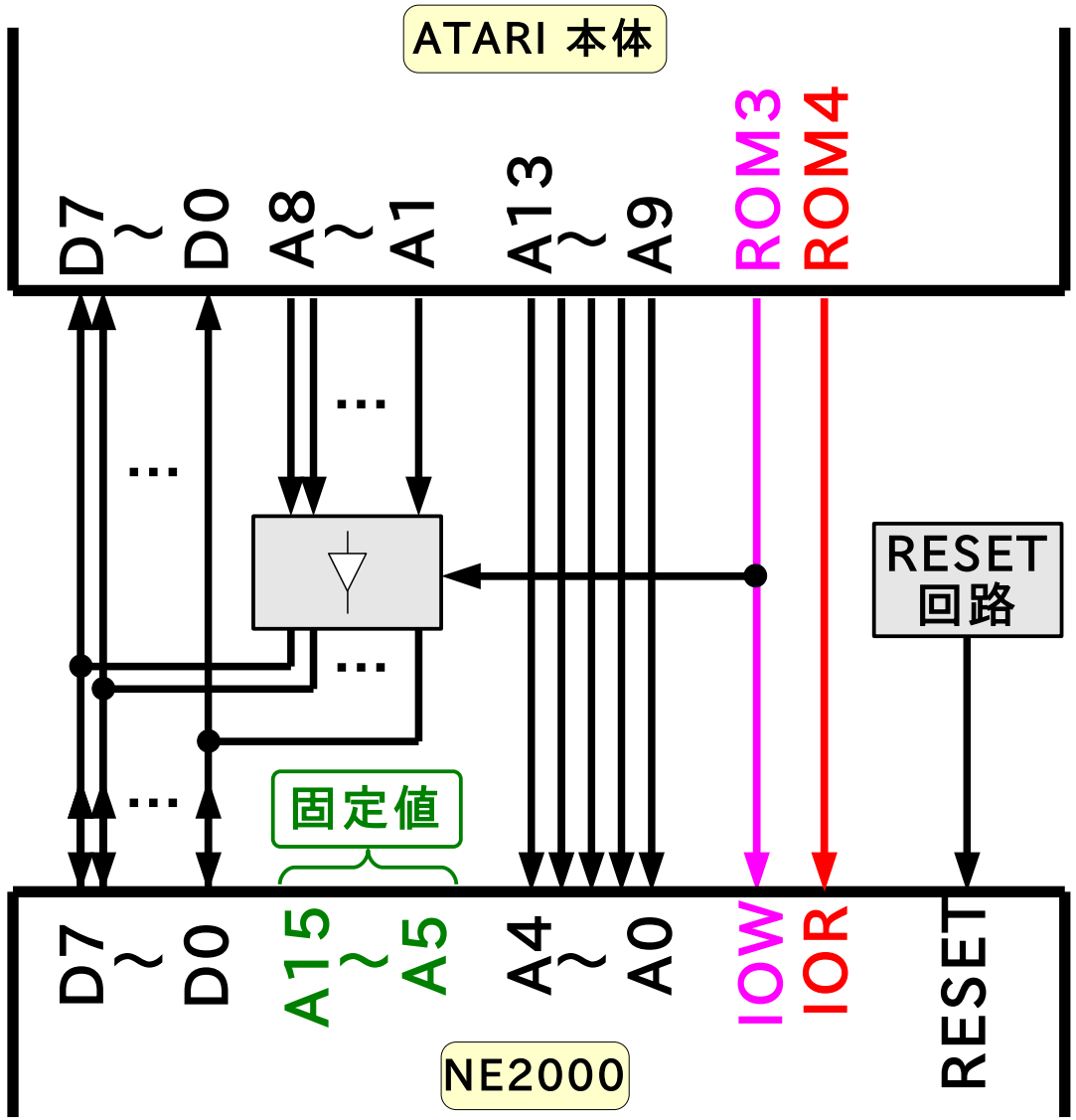
…なんか上から下に行く信号が
たくさんあるよね

How to connect NE2000 to ROM slot? (2)



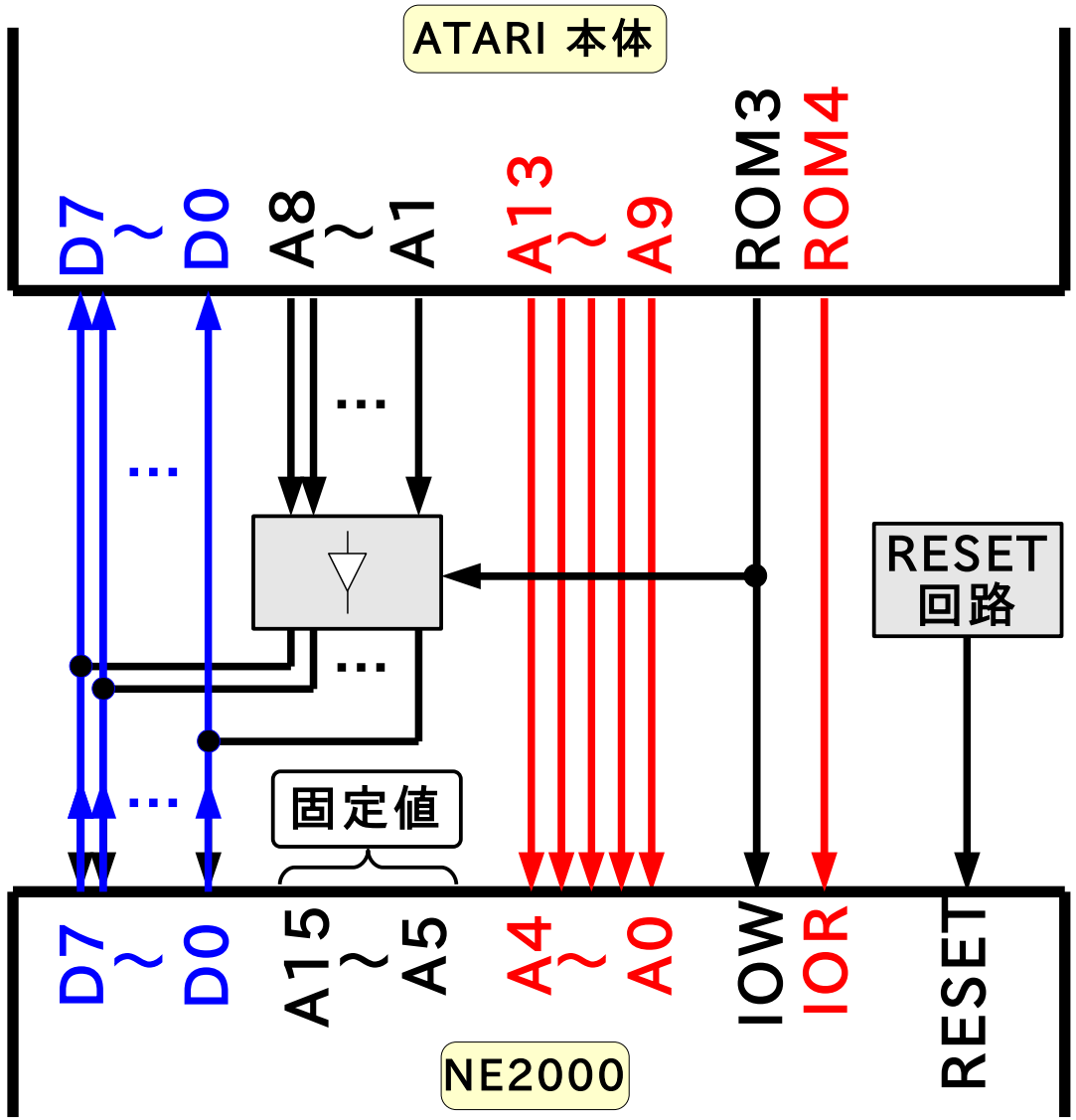


How to connect NE2000 to ROM slot? (3)



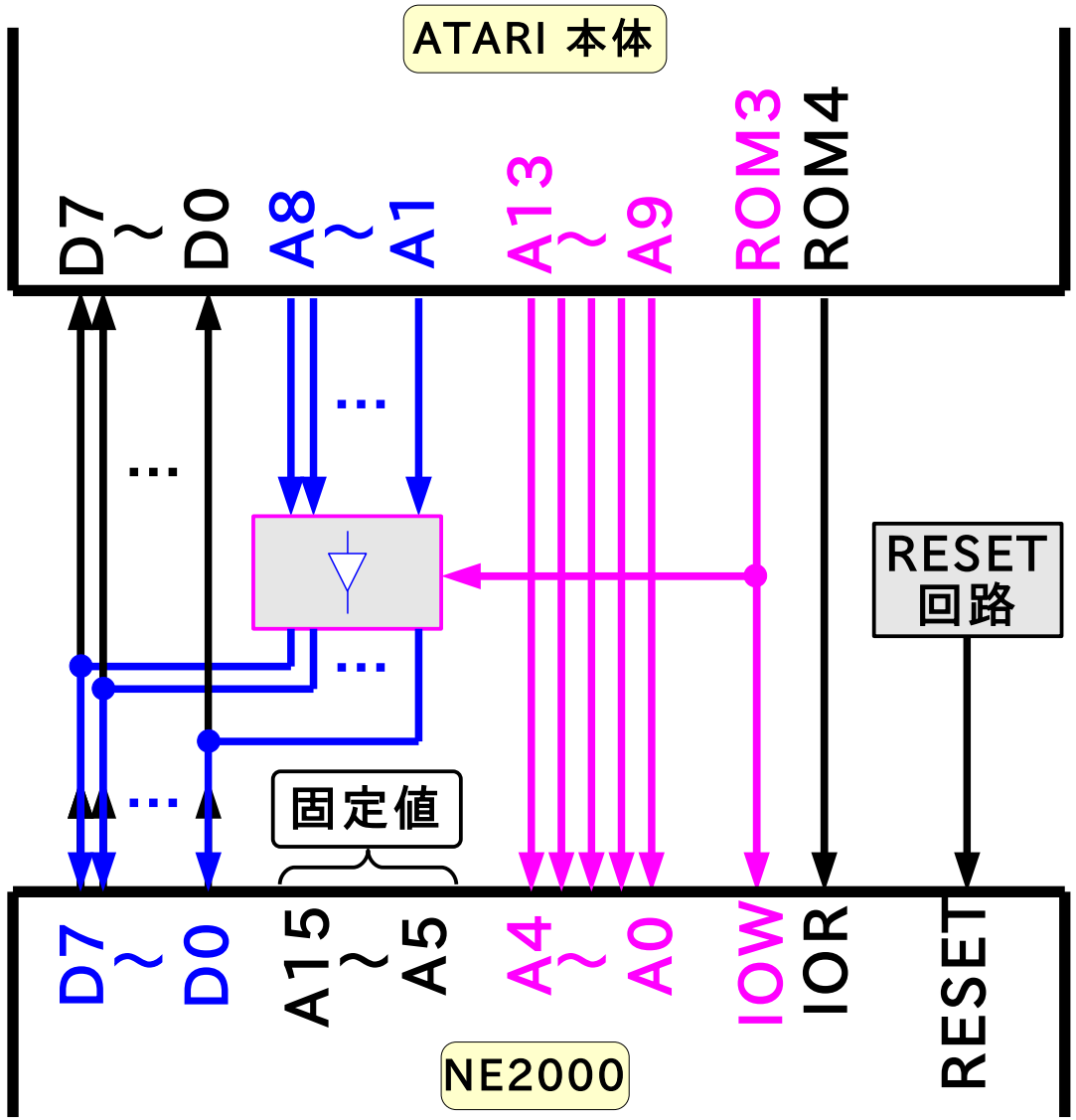
- ROM3 ⇒ IOW
  ⇒ 0xFBxxxx への
  アクセスで書き込み
  （本体⇒NE2000へ入力）
- ROM4 ⇒ IOR
  ⇒ 0xFAxxxx への
  アクセスで読み出し
  （NE2000⇒本体へ出力）
- ROM3,4はEtherNEC専用
  として、ボード選択は固定

How to connect NE2000 to ROM slot? (4)



- A13～A9 ⇒ A4～A0
  NE2000 レジスタ選択
  - 0xFA**00**xx or 0xFA**01**xx で
    NE2000 レジスタ0 をリード
  - 0xFA**02**xx or 0xFA**03**xx で
    NE2000 レジスタ1 をリード
  - 0xFA**3E**xx or 0xFA**3F**xx で
    NE2000 レジスタ31をリード
- D7～0 直結
  - リードのときはデータは
    D0～D7 でそのまま読み出し

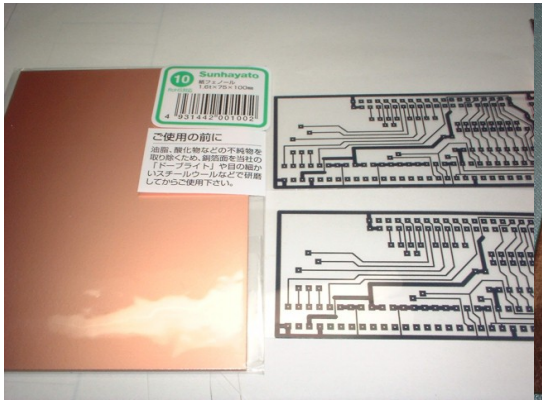
How to connect NE2000 to ROM slot? (5)



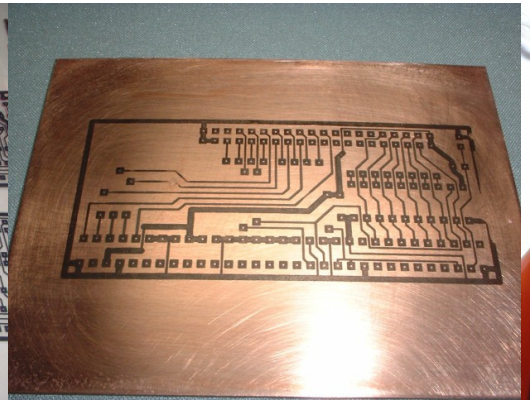
- ライト時 A8～1⇒D7～0：
  下位アドレスでライト値指定
  - 0xFB**0002** へのアクセスで
    レジスタ0 に 0x01 をライト
  - 0xFB**02AA** へのアクセスで
    レジスタ1 に 0x55 をライト
  - 0xFB**3FFE** へのアクセスで
    レジスタ31に 0xFF をライト
- NE2000 8bitモード使用
- 割り込みはないのでポーリング
- 簡易RESET発生回路

EtherNEC H/W作成

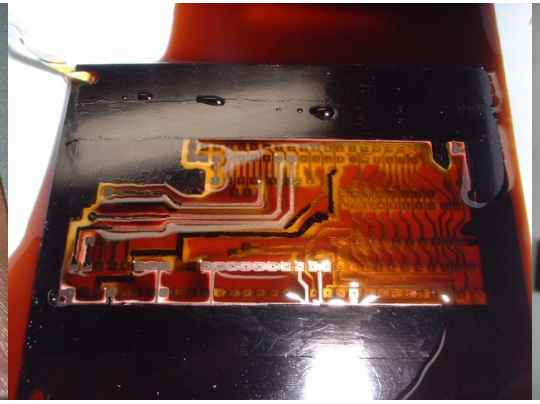
……仕様をだいたい把握したところで、通販は送料が高いので現物を作成



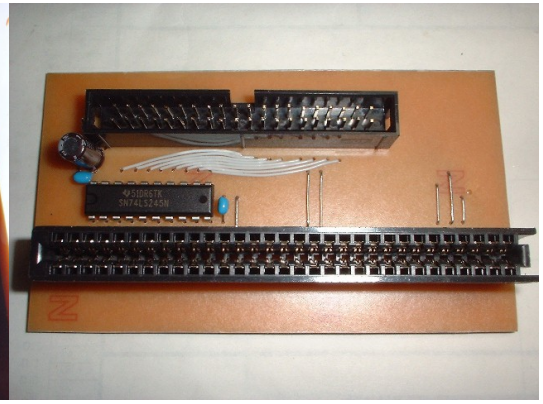
生基板とパターン図



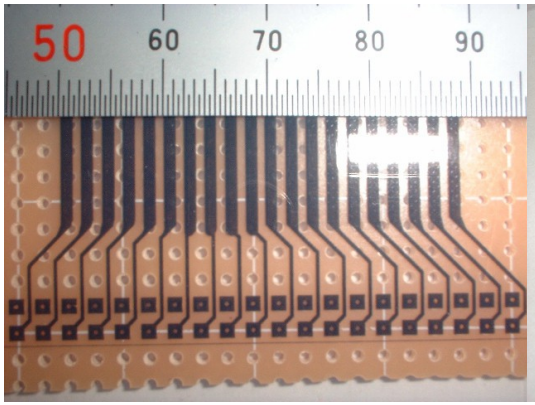
パターン転写



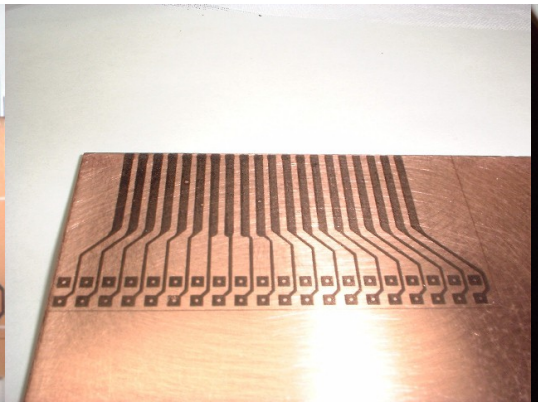
エッチング



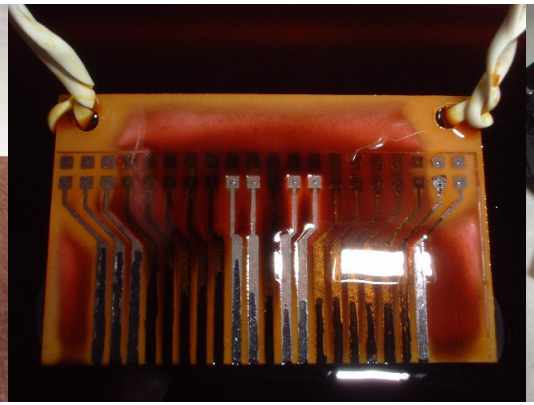
部品取り付け



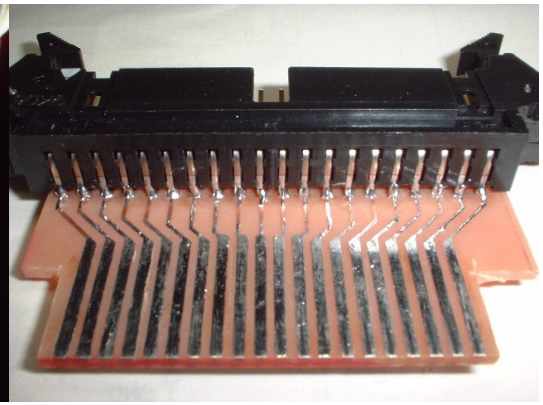
ピッチ(2mm)確認



パターン転写（両面）



エッチング



コネクタ半田づけ

工作その他の全体作成記は togetter まとめ参照 http://togetter.com/li/121717

NetBSD MI デバイスドライバ

- **MI (Machine Independent) NE2000ドライバ**
  - 「32個分のレジスタに適当に読み書きしてパケット送受信」の部分が書かれたソース

    `sys/dev/ic/ne2000.c sys/dev/ic/dp8390.c` 等

  - ISA NE2000, X680x0用Neputune-X などはいずれもこの MI NE2000 ドライバを使用

  - EtherNEC でも
    「どういう手順でどのレジスタをアクセスするか」は同じ。

    「どんな手段で各レジスタにアクセスするか」という
    ハードウェア的な接続が違うだけ。

bus_space API の目的

- **「異なるマシン、異なる接続、だけど同じデバイス」**
  ならば MIデバイスドライバのソースは1つで済ませたい

- **「異なるバス接続だけど同一マシン上の同じデバイス」**
  ならば MIデバイスドライバのバイナリも1つで済ませたい

⇒ 機種やバス接続により異なる部分だけを MI から分離して
MD (Machine Dependent) 部に記述する必要がある。

……この目的の実現のための手段が bus_space API

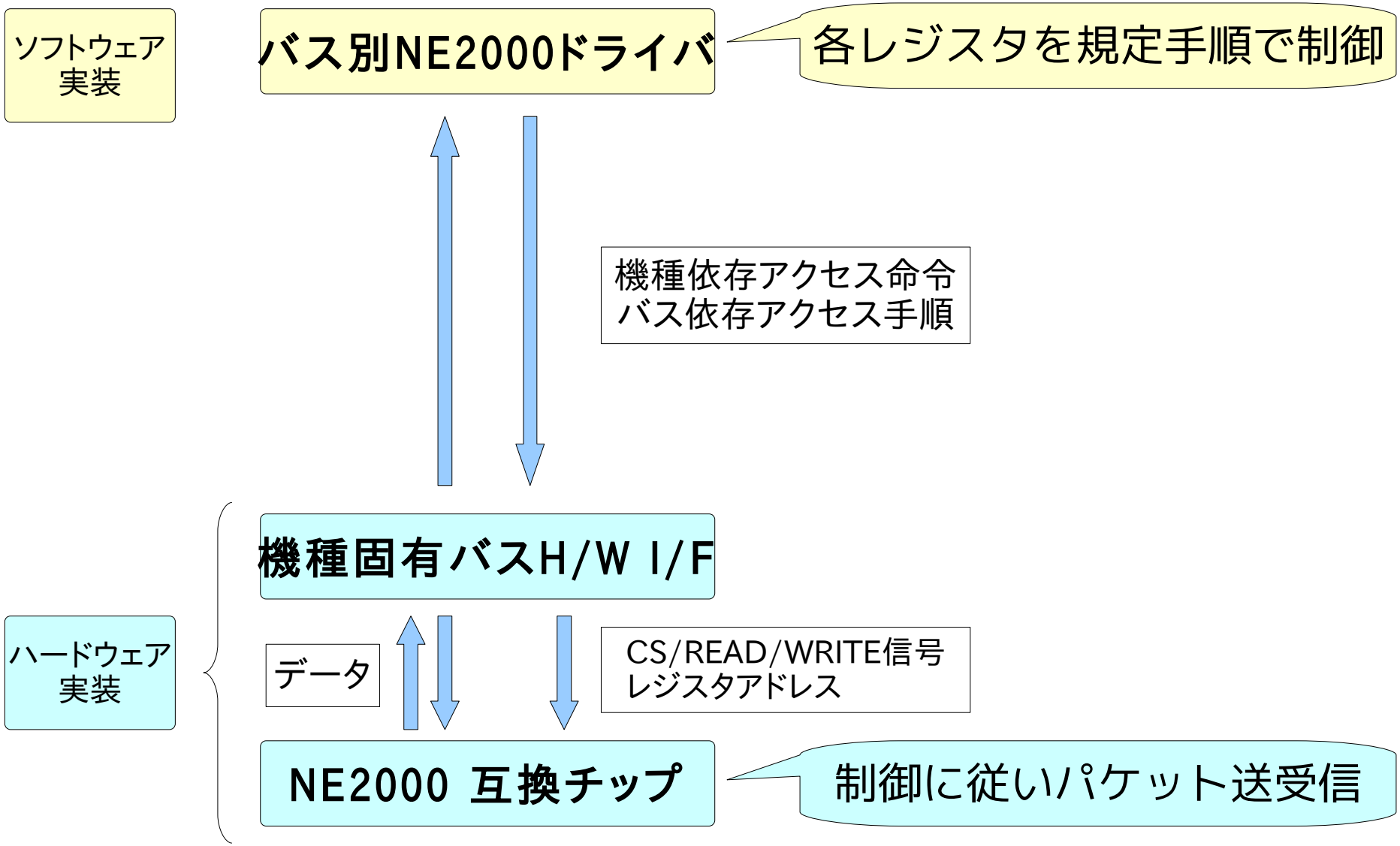
詳細は予習テキスト参照
http://www.ceres.dti.ne.jp/tsutsui/netbsd/doc/bus_space2.html

bus_space API の考え方

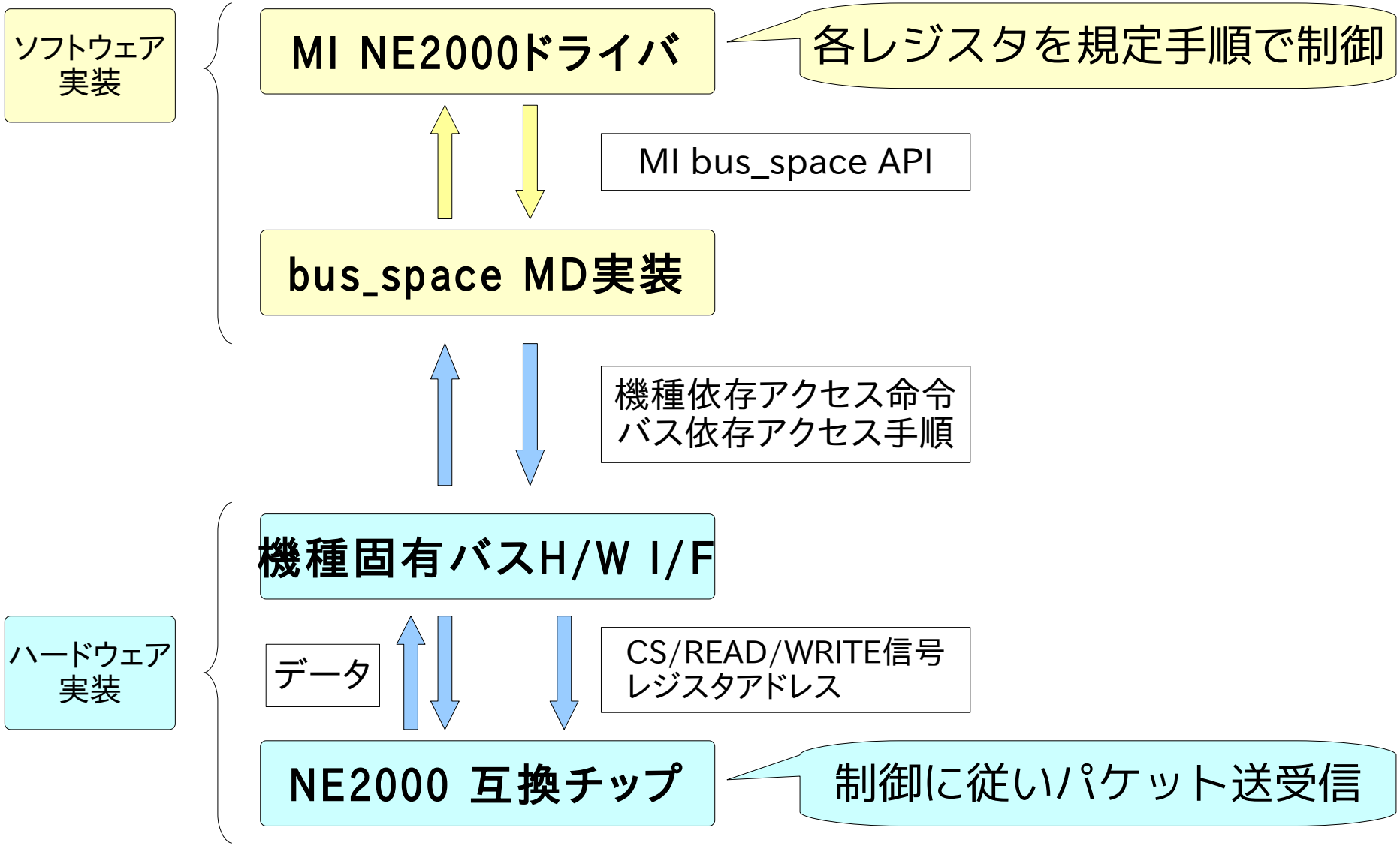
- バス,デバイス依存実装 ⇒ MIドライバ記述から「隠す」必要アリ
  - デバイスのアクセス方法
    - x86なら **`inb/outb`** とか RISC CPUなら **`load/store`** とか
    - データ並び順やアドレス指定など特殊手順がある場合も
  - デバイス毎の情報
    - ISA なら アドレス 0x280 なのか 0x300 なのかとか
    - 同じバスでも個別処理がいる場合はその例外の内容とか
- 「どのレジスタをアクセスするか」は共通 ⇒ MIドライバに記述

参考URL 曽田さんの解説メール
http://www.clave.gr.jp/ml/bsd-nomads/200004/msg00159.html

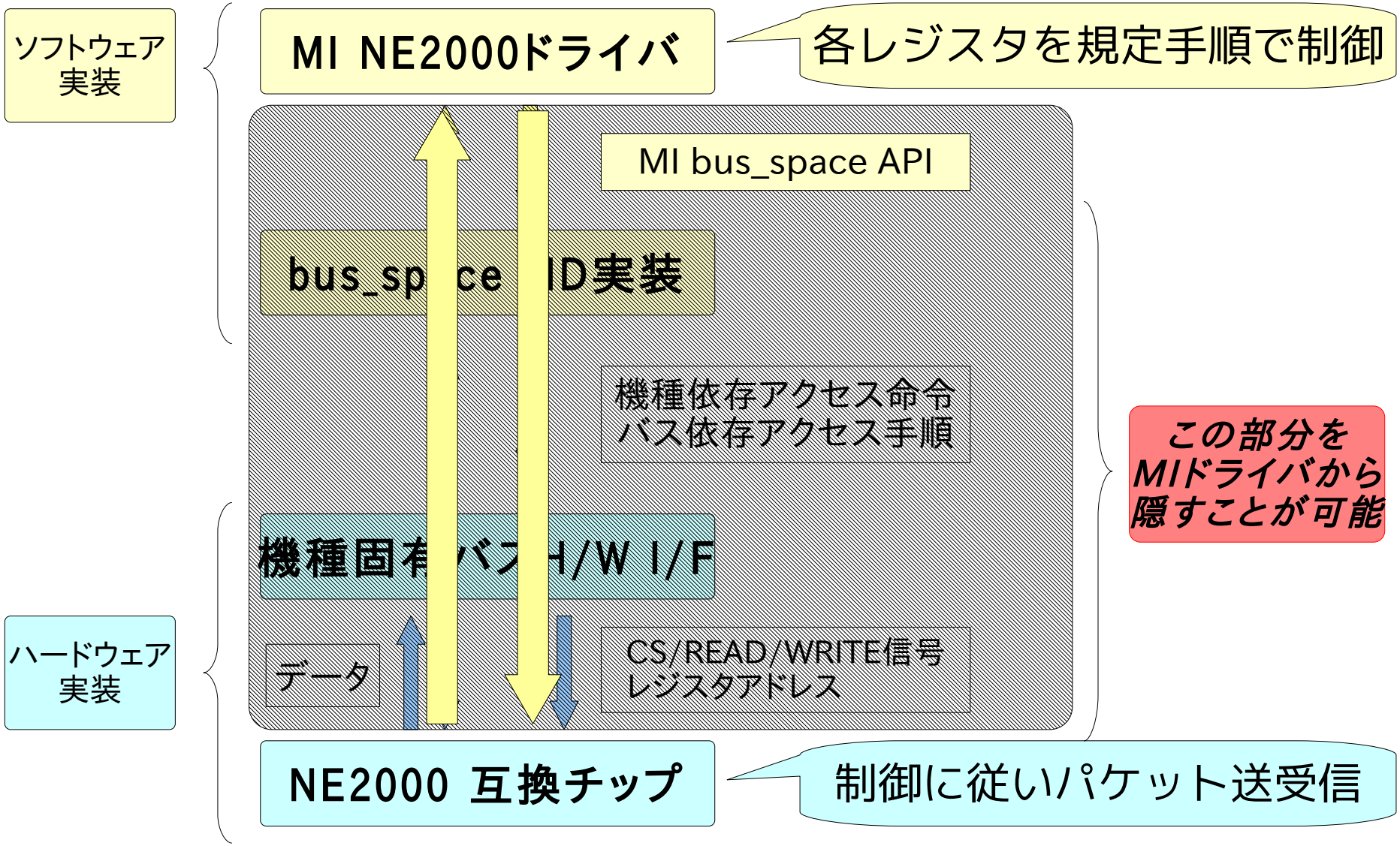
旧来のデバイスドライバ実装



bus_space API を使った実装



bus_space API を使った実装



bus_space API のアクセス記述

- アクセス時には以下の3つをセットで指定する
  (1) bus space tag　(MD実装隠蔽用 謎変数 (opaque) その1)
  (2) bus space handle　(MD実装隠蔽用 謎変数 (opaque) その2)
  (3) オフセット(どのレジスタをアクセスするか)および 書き込み値

```
foo_ops(struct foo_softc *sc)
{
        bus_space_tag_t    bt = sc->sc_bst;
        bus_space_handle_t bh = sc->sc_bsh;
        uint8_t            rval

        bus_space_write_1(bt, bh, REG1, 0);
        rval = bus_space_read_1(bt, bh, REG2);
}
```

bus space tag と bus space handle (1)

bus space tag って何? bus space handle って何?

(1) MIドライバを書く人向け

⇒ 知らないほうが幸せ。

MI APIが返してきた値を何も考えずそのまま使用せよ。

(2) 既存バスの新デバイスでMIドライバを使いたい人向け

⇒ 知らなくてもなんとかなる。

が、なんとなくのイメージはないと困るかも。

- bus space tag: なんとなくバス毎で固有っぽい何か。
  - 主にアクセス方法など。上位層のバスドライバから渡される。
- bus space handle: なんとなくデバイス毎で異なる何か。
  - 主にアドレスなど。bus_space_map()で設定する。

bus space tag と bus space handle (2)

bus space tag って何? bus space handle って何?

(3) 新たなバスとデバイスを追加する人向け

⇒ MIドライバから隠すために押し込まれたMD構造の
有象無象。健闘を祈る(おい

- アクセス方法、アドレス、その他H/Wアクセス用情報
- どういう構造で、どこに何を実装、という決まりは無い。
  - MIドライバ利用者の安息のためなら正直何でもあり。
- 簡単なバスなら簡単。複雑なバスやデバイスなら複雑。
  ハードウェアの実装次第。

NetBSD/atari での bus_space実装

- NetBSD/atari 実装の bus space tag の中身
  - 以下を含む **構造体へのポインタ**。
    - 各 bus_space API の read/writeアクセスAPI実装への関数ポインタ
    - バス別 ベースアドレス値
    - バス幅別 アドレス調整値 (stride値)

- NetBSD/atari 実装の bus space handle の中身
  - デバイスのアクセスに使用する **仮想アドレス値**。
    または オフセット値。

⇒ わりと柔軟な記述ができる構造の実装になっているので、
EtherNEC用実装に対しては、そのまま流用でなんとかなりそう

EtherNEC で必要な bus_space実装

- **EtherNECアクセス用 bus space tag**
  - EtherNECはカートリッジスロットという独立したバスもどきなので bus space tag のMD実装である構造体の実体を用意

- **~~手抜き~~巧妙ハード実装に対応したアクセス方法記述**
  - 同じレジスタでも read と write で違うアドレス
  - レジスタ選択用オフセットは上位アドレス A13～A9
  - 書き込みもリードアクセスで、下位アドレスで書き込み値を指定

  ⇒ 読み書き用の関数は関数ポインタで個別に指定できるので、
  それぞれアクセスAPI関数を書いて bus space tag に設定

EtherNEC での bus_space アクセス実装 (1)

(1) bus_space_read_1(): レジスタ読み出しアクセス部分

```
/* CPU address lines A13-A9 are connected to ISA A4-A0 */
#define ETHERNEC_PORT_STRIDE  9

static uint8_t
ethernec_bus_space_read_1(bus_space_tag_t bt, bus_space_handle_t bh,
    bus_size_t reg)
{
    volatile uint8_t *ba;

    ba = (volatile uint8_t *)(bh + (reg << ETHERNEC_PORT_STRIDE));

    return *ba;
}
```

詳細はソースコード参照 http://nxr.NetBSD.org/xref/src/sys/arch/atari/dev/if_ne_mb.c

EtherNEC での bus_space アクセス実装 (2)

(2) bus_space_write_1(): レジスタ書き込みアクセス部分

```
/* Use A8-A1 lines to specify write data (no A0 but UDS/LDS on m68k) */
#define ETHERNEC_WR_ADDR_SHIFT   1

static void
ethernec_bus_space_write_1(bus_space_tag_t bt, bus_space_handle_t bh,
    bus_size_t reg, uint8_t val)
{
    volatile uint8_t *wport;

    wport = (volatile uint8_t *)(bh +
        bt->stride + (reg << ETHERNEC_PORT_STRIDE));
    wport += (u_int)val << ETHERNEC_WR_ADDR_SHIFT;

    (void)*wport;
}
```

詳細はソースコード参照 http://nxr.NetBSD.org/xref/src/sys/arch/atari/dev/if_ne_mb.c

EtherNEC での bus_space アクセス実装 (3)

(3) MIドライバからの bus_space_write_1() API 呼び出し

sys/dev/ic/ne2000.c:

```
  bus_space_write_1(asict, asich, NE2000_ASIC_DATA, 0);
```

sys/arch/atari/include/bus.h:

```
#define bus_space_write_1(t, h, o, v) __abs_ws(1,(t),(h),(o),(v))
struct atari_bus_space {
 :
  /* write (single) */
  void  (*abs_w_1)(bus_space_tag_t, bus_space_handle_t, bus_size_t, uint8_t);
 :
};
```

sys/arch/atari/dev/if_ne_mb.c:

```
ethernec_init_bus_space_tag(bus_space_tag_t en_t)
{
 :
  en_t->abs_w_1   = ethernec_bus_space_write_1;
}
```

詳細はソースコード参照 http://nxr.NetBSD.org/xref/src/sys/arch/atari/include/bus.h

bus space まとめ (1)

- **MIデバイスドライバ**
  - 最初に誰かが一つMIドライバを書けばあとのみんなが幸せ
    …… EtherNEC も SMC_TT もドライバは一晩で書いて動作
- **MI実装実現手段**
  - bus space tag と bus space handle という謎opaque変数を使用
  - 普通に使うなら謎変数は知らないままでも幸せ
    ……先人が苦労して隠してくれてます

⇒ 新たなマシンやバスの実装でMIドライバを流用したいときは
これら謎変数実装のパンドラの箱を開ける必要あり。

bus space まとめ (2)

- 「**MIドライバをみんなで使えるようにする**」のが本題。
  - MIドライバを変更するとみんなに影響する。
    ⇒ 変なことを書くと怒られる。作ってしまった APIは変えられない。
  - MD側の個別実装を変更してもその人以外は困らない。
    ⇒ あとからでも修正が可能。

- bus_space の MD側実装はある意味**何でもアリ**。
  - MI側の「きれいな実装」とは意識を変えて読解/実装が必要
  - 「きれいな設計」も大事だが、汚いハード実装に対する妥協も重要

    ……ハードは **製造コスト** に直結するが、「ソフトは**タダ**」という現実 (´・ω・`)

bus space まとめ (3)

- 某FreeBSDコミッタ氏の過去のMLでの問題発言

  “Busspace is a joke, I'm perfectly fine with calling those macros in[bwl]/out[bwl] like they should be :)”

  - 正直、名前だけなら bus_space() でも inb() でもなんでもいい。
  - でも、x86以外の命令や、EtherNECみたいな変態H/Wの存在は知っておくべき。世の中そんなに簡単じゃない。

    ……「Machine Independent」という体を表す名前のほうがいいよね

終わり

- 長時間ご清聴ありがとうございました _o_
- これであなたも bus_space がわかった気になったはず!?
- bus_dma はもっとややこしいのでまた機会があれば……